

# 无标题

初识RabbitMq:

rabbitmq是一个开源的消息代理和队列服务器，通过普通的协议(Amqp协议)来完成不同应用之间的数据共享

(消费生产和消费者 可以跨语言平台)

rabbitmq是通过erlang语言来开发的基于amqp协议

一：各大互联网公司为什么选择Rabbitmq

- 1)比如滴滴，美团，携程，去哪儿
- 2)开源，性能好，稳定性保证，
- 3)提供了消息的可靠性投递（confirm），返回模式
- 4)与spring amqp 整合和完美，提供丰富的api
- 5)集群模式十分丰富(HA模式 镜像队列模型)
- 6)保证数据不丢失的情况下，保证很好的性能

二：Rabbitmq高性能是如何做到的

- 1) 使用的语言是erlang语言(通常使用到交互机上)，erlang的语言的性能能与原生socket的延迟效果.
- 2)消息入队的延时已经消息的消费的响应很快

三:什么是AMQP协议(Advanced message queue protocol) 高级消息队列协议

1:)是一个二进制协议，

2) amqp 是一个应用层协议的规范（定义了很多规范），可以有很多不同的消息中间件产品（需要遵循该规范）

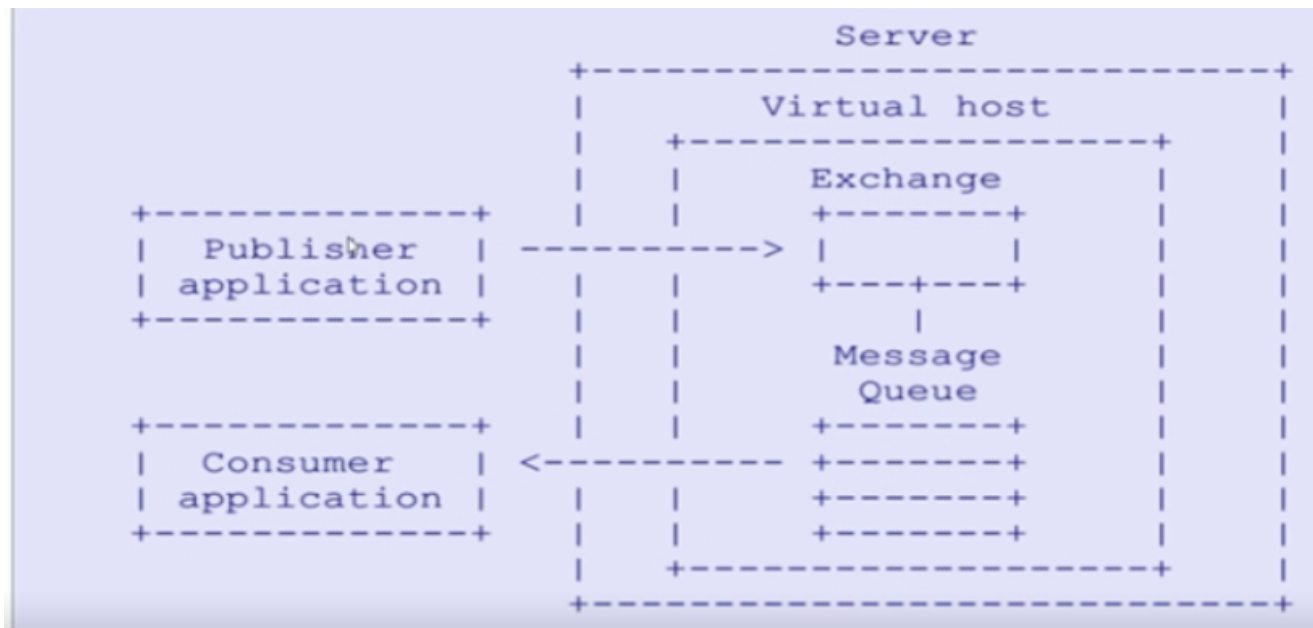
server: 是消息队列节点

virtual host:虚拟注解

exchange 交换机(消息投递到交换机上)

message queue （被消费者监听消费）

交互机和队列是有一个绑定的关系

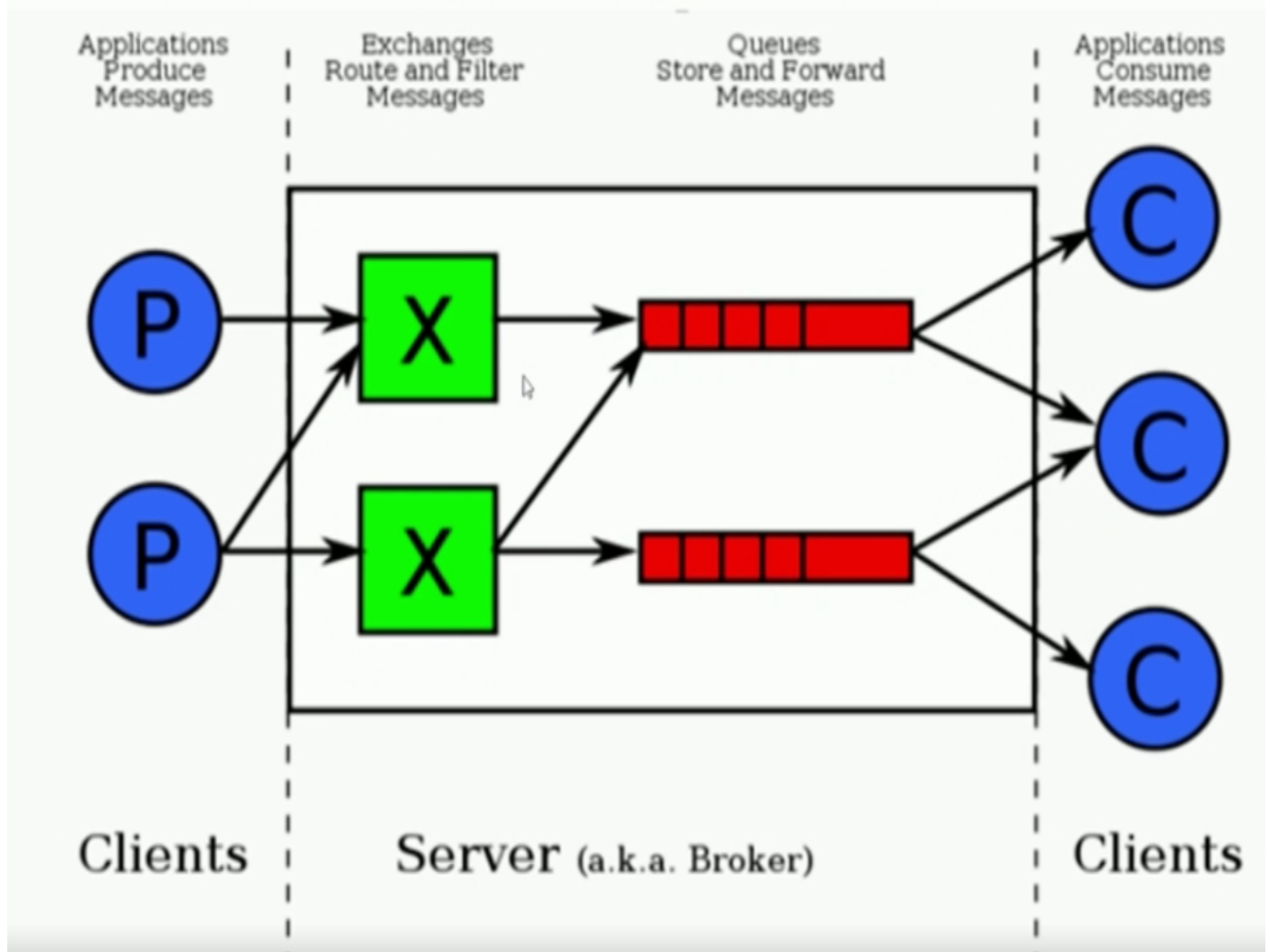


#### 四:AMQP的核心概念

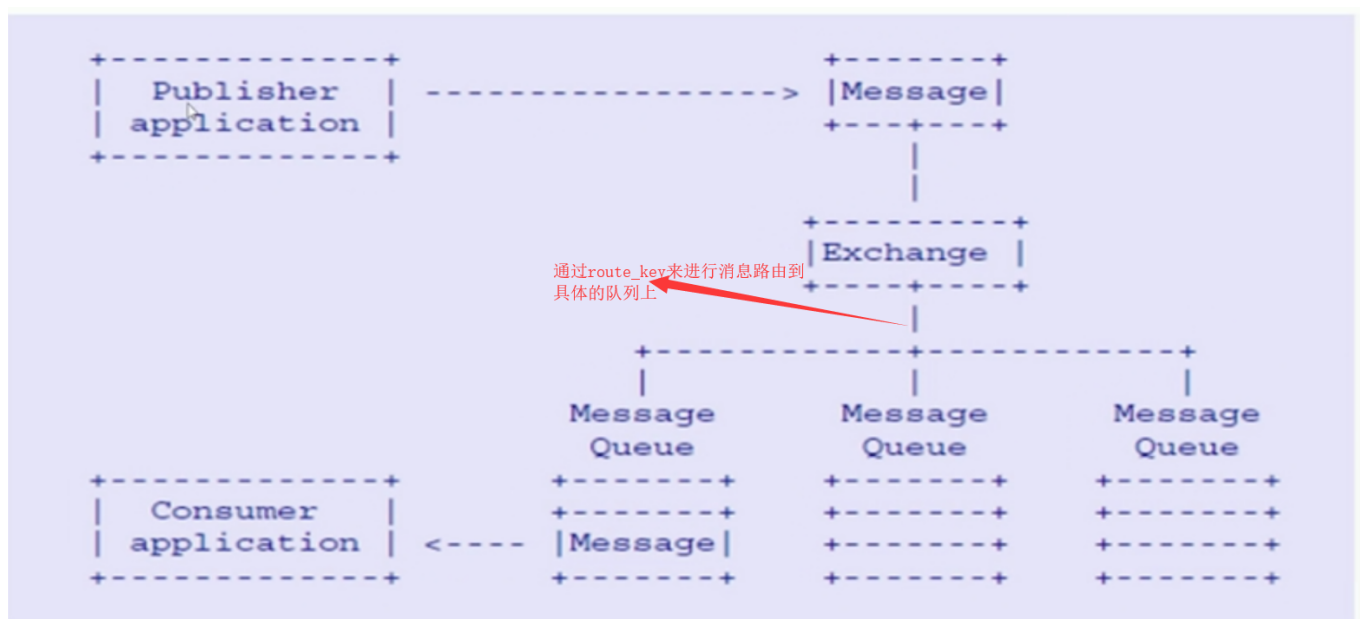
- 1:server :又称为broker, 接受客户端连接, 实现amqp实体服务
- 2:Connection: 连接,应用程序与broker建立网络连接
- 3: channel: 网络通道, 几乎所有的操作都是在channel中进行的, 是进行消息对象的通道, 客户端可以建立多个通道, 每一个channel表示一个会话任务
- 4:Message: 服务器和应用程序之间传递数据的载体, 有properties (消息属性,用来修饰消息,比如消息的优先级,延时投递) 和Body (消息体)
- 5:virtual host(虚拟主机): 是一个逻辑概念,最上层的消息路由, 一个虚拟主机中可以包含多个exchange 和 queue 但是一个虚拟主机中不能有名称相同的exchange 和queue
- 6:exchange 交换机: 消息直接投递到交换机上, 然后交换机根据消息的路由key 来路由到对应绑定的队列上
- 7:binding: 绑定 exchange 与queue的虚拟连接,binding中可以包含route\_key
- 8: route\_key 路由key , 他的作用是在交换机上通过route\_key来把消息路由到哪个队列上
- 9:queue: 队列, 用于来保存消息的载体, 有消费者监听, 然后消费消息

#### 五:Rabbitmq的整体架构模型

# RabbitMQ的整体架构是什么样子的?



## 六:rabbitmq的消息是如何流转的



## 七:rabbitmq的安装和使用

### 1)首先下载对应的安装包

wget [www.rabbitmq.com/releases/erlang/erlang-18.3-1.el7.centos.x86\\_64.rpm](http://www.rabbitmq.com/releases/erlang/erlang-18.3-1.el7.centos.x86_64.rpm) 安装rabbitmq 的环境(erlang语言环境)

wget [http://repo.iotti.biz/CentOS/7/x86\\_64/socat-1.7.3.2-5.el7.linux.x86\\_64.rpm](http://repo.iotti.biz/CentOS/7/x86_64/socat-1.7.3.2-5.el7.linux.x86_64.rpm) 下载rabbitmq依赖的环境包

wget [www.rabbitmq.com/releases/rabbitmq-server/v3.6.5/rabbitmq-server-3.6.5-1.noarch.rpm](http://www.rabbitmq.com/releases/rabbitmq-server/v3.6.5/rabbitmq-server-3.6.5-1.noarch.rpm) 下载rabbitmq



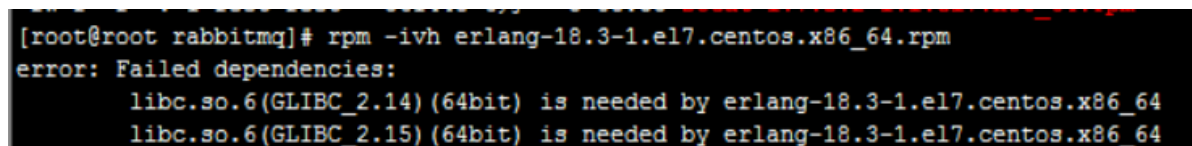
名称	大小	类型	修改时间
..			
erlang-18.3-1.el7.centos.x86_64.rpm	17.50MB	快压 RPM...	2018/6/5, 8:33
rabbitmq-server-3.6.5-1.noarch.rpm	5.26MB	快压 RPM...	2018/6/5, 8:33
socat-1.7.3.2-1.1.el7.x86_64.rpm	353KB	快压 RPM...	2018/6/5, 8:33

### 2)首先执行安装erlang语言的环境( cd /usr/local/rabbitmq 目录下)

#### 第一步:先安装gcc环境

```
yum install build-essential openssl openssl-devel unixODBC unixODBC-devel make gcc gcc-c++  
kernel-devel m4 ncurses-devel tk tc xz
```

第二步: rpm -ivh erlang-18.3-1.el7.centos.x86\_64.rpm 安装erlang语言的环境,若提示出现下面错误



```
[root@root rabbitmq]# rpm -ivh erlang-18.3-1.el7.centos.x86_64.rpm  
error: Failed dependencies:  
    libcrypto.so.6(GLIBC_2.14) (64bit) is needed by erlang-18.3-1.el7.centos.x86_64  
    libcrypto.so.6(GLIBC_2.15) (64bit) is needed by erlang-18.3-1.el7.centos.x86_64
```

那么说明Glibc的版本过低 我们需要安装GLIBC\_2.14

2.1)先查看我们本地的GLIBC的版本 strings /lib64/libc.so.6 | grep GLIBC 查看版本 发现最高版本才2.12

GLIBC\_2.2.5  
GLIBC\_2.2.6  
GLIBC\_2.3  
GLIBC\_2.3.2  
GLIBC\_2.3.3  
GLIBC\_2.3.4  
GLIBC\_2.4  
GLIBC\_2.5  
GLIBC\_2.6

GLIBC\_2.7  
GLIBC\_2.8  
GLIBC\_2.9  
GLIBC\_2.10  
GLIBC\_2.11  
GLIBC\_2.12  
GLIBC\_PRIVATE  
-----

## 2.2) 下载 [glibc-2.14.tar.gz](http://ftp.gnu.org/gnu/glibc/glibc-2.14.tar.gz)

a: 下载 `wget http://ftp.gnu.org/gnu/glibc/glibc-2.14.tar.gz`

b: 解压 `tar xf glibc-2.14.tar.gz`

C: 进入解压目录 `cd glibc-2.14`

d: 创建build目录 `mkdir build`

e: 配置目录: `./configure --prefix=/opt/glibc-2.14 --disable-profile --enable-add-ons --with-headers=/usr/include --with-binutils=/usr/bin`

f: 编译 `make`

g: 安装: `make install`

\*\*\*\*\*此时安装成功glibc2.14 但是通过命令(strings /lib64/libc.so.6 |grep GLIBC)看不到2.14版本

### 解决方案:

安装完成后, 建立软链指向glibc-2.14, 执行如下命令:

1. `$ rm -rf /lib64/libc.so.6 // 先删除先前的libc.so.6软链`
2. `$ ln -s /opt/glibc-2.14/lib/libc-2.14.so /lib64/libc.so.6`

删除libc.so.6之后可能导致系统命令不可用的情况, 可使用如下方法解决:

```
$ LD_PRELOAD=/opt/glibc-2.14/lib/libc-2.14.so ln -s /opt/glibc-2.14/lib/libc-2.14.so  
/lib64/libc.so.6
```

-----  
欢迎到2.12版本

```
$ LD_PRELOAD=/lib64/libc-2.12.so ln -s /lib64/libc-2.12.so /lib64/libc.so.6 // libc-2.12.so 此项是系  
统升级前的版本
```

第三步:安装socat 包: rpm -ivh socat-1.7.3.2-5.el7.linux.x86\_64.rpm

第四步: 安装 rabbitmq: rpm -ivh [rabbitmq-server-3.6.5-1.noarch.rpm](#)

第五步:简单配置rabbitmq

vim /usr/lib/rabbitmq/lib/rabbitmq\_server-3.6.5/ebin/rabbit.app

比如修改密码、配置等等, 例如: loopback\_users 中的 <<"guest">>,只保留guest

第六步:rabbitmq的起停服务 进入到rabbitmq目录下

/usr/lib/rabbitmq/lib 运行./rabbitmq-server start(./rabbitmqctl start\_app) 停止服务  
./rabbitmq-server stop(./rabbitmqctl stop\_app)

## 八:命令行和管控台

开启管控台插件 ./rabbitmq-plugins rabbitmq\_management 来开启管控台

测试连接: <http://ip:15672/>(来访问) 用户名密码 guest/guest

The screenshot shows the RabbitMQ Management UI. At the top, there's a navigation bar with tabs: Overview, Connections, Channels, Exchanges, Queues, and Admin. The Overview tab is selected. Below the navigation bar, there's a section for 'Totals' with various metrics: Queued messages (chart: last minute) (?), Currently idle, Message rates (chart: last minute) (?), Currently idle, and Global counts (?). Below this, there are buttons for Connections: 0, Channels: 0, Exchanges: 8, Queues: 0, and Consumers: 0. There's also a 'Node' section with a table of metrics: File descriptors (?), Socket descriptors (?), Erlang processes, Memory, Disk space, Rates mode, and Info. The table shows values: 52, 0, 226, 50MB, 35GB, basic, and Disc 1 Stats. Below the table, there's a 'Paths' section with a list of files and their locations: Config file, Database directory, Log file, and SASL log file. At the bottom, there's a 'Ports and contexts' section.

File descriptors (?)	Socket descriptors (?)	Erlang processes	Memory	Disk space	Rates mode	Info
52	0	226	50MB	35GB	basic	Disc 1 Stats

Path	Value
Config file	/etc/rabbitmq/rabbitmq.config (not found)
Database directory	/var/lib/rabbitmq/mnesia/rabbit@localhost
Log file	/var/log/rabbitmq/rabbit@localhost.log
SASL log file	/var/log/rabbitmq/rabbit@localhost-sasl.log

### 8.1)管理控制台命令:

#### a.起停服务命令

**启动服务** rabbitmqctl start\_app

**停止服务** rabbitmqctl stop\_app

**查看服务状态** rabbitmqctl status

#### b:用户操作命令

**添加用户** rabbitmqctl add\_user root root

**查询所有用户** rabbitmqctl list\_users

**删除用户** rabbitmqctl delete\_user root

**清除用户权限** rabbitmqctl clear\_permissions -p vhostpath username

**列出用户权限** rabbitmqctl list\_user\_permissions username

**修改密码** rabbitmqctl change\_password 用户名 新密码

**设置用户权限** rabbitmqctl set\_permissions\_p vhost 用户名 "." "\*" "\*" "\*" "

#### c:虚拟主机操作

rabbitmqctl add\_vhost /cloudmall **增加一个虚拟主机**

rabbitmqctl list\_vhosts; **查看所有的虚拟主机**

rabbitmqctl list\_permissions -p /cloudmall **查看虚拟主机的权限**

rabbitmqctl delete\_vhost /cloudmall **删除虚拟主机**

#### D:操作队列命令

rabbitmqctl list\_queues **查询所有队列**

rabbitmqctl -p vhostpath purge\_queue blue **清除队列消息**

#### E:高级命令

rabbitmqctl reset **移除所有数据** 该命令需要在 rabbitmqctl stop\_app命令之后才执行(也就是说 在服务停止后)

rabbitmqctl join\_cluster <cluster\_node> [--ram] **组成集群命令**

rabbitmqctl cluster\_status **查看集群状态**

rabbitmqctl change\_cluster\_node\_type dist|ram **修改集群节点存储数据模式**

rabbitmqctl forget\_cluster\_node [--offline]忘记节点（摘除节点）

rabbitmqctl rename\_cluster\_node oldnode1 newnode1 oldnode2 newnode2 修改节点名称

## 九:消费者 生产者模型(使用java连接mq)

### A:生产者

```
public static void main(String[] args) throws IOException, TimeoutException {
    //创建一个连接工厂
    ConnectionFactory connectionFactory = new ConnectionFactory();
    //设置连接工厂属性
    connectionFactory.setHost("47.104.128.12");
    connectionFactory.setPort(5672);
    connectionFactory.setVirtualHost("/");
    //创建连接
    Connection connection = connectionFactory.newConnection();
    //创建channel
    Channel channel = connection.createChannel();
    //通过channel操作mq
    for(int i=0;i<10;i++) {
        String msg = "生产者生产的第" + (i+1) + "条消息";
        //若exchange 为空那么消息可以发送到任何队列上,只要routKey 与队列名称相同就会发送上去。
        channel.basicPublish("", "qk001", null, msg.getBytes());
    }
    //关闭资源
    channel.close();
    connection.close();
}
```

### B: 消费者:

```
public static void main(String[] args) throws IOException, TimeoutException, InterruptedException {
    //创建一个连接工厂
    ConnectionFactory connectionFactory = new ConnectionFactory();
    //设置连接工厂属性
    connectionFactory.setHost("47.104.128.12");
    connectionFactory.setPort(5672);
    connectionFactory.setVirtualHost("/");
    //创建连接
    Connection connection = connectionFactory.newConnection();
    //创建连接
    Channel channel = connection.createChannel();
    //声明一个队列
    /**
```



```

* 第一个参数:队列名称
* 第二个参数: 队列消息可以持久化
* 第三个参数:表示该队列只能被一个channel操作
* 第四个参数:若队列没有绑定交换机，那么消息会被删除
*/
channel.queueDeclare("qk001",true,false,false,null);
//创建一个消费者
QueueingConsumer queueingConsumer = new QueueingConsumer(channel);
//设置channel
channel.basicConsume("qk001",true,queueingConsumer);
//消费消息
while (true) {
    Delivery delivery = queueingConsumer.nextDelivery(); //阻塞 若设置参数就会在指定时间结束
    String msg = new String(delivery.getBody());
    System.out.println("消费消息:"+msg);
}
}

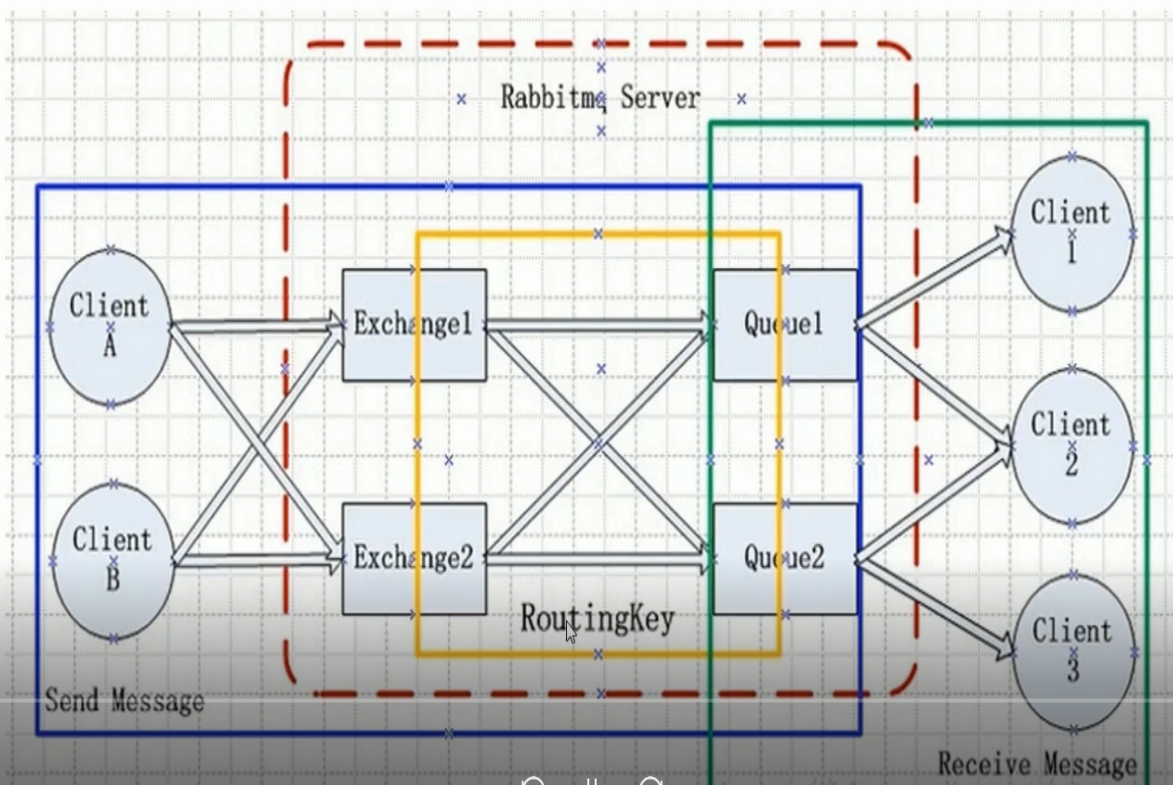
```

## 十:Rabbitmq交换机详解

1: 作用:接受生产者的消息，然后根据路由键 把消息投递到跟交换机绑定的对应的队列上

# Exchange 交换机

✓ Exchange: 接收消息，并根据路由键转发消息所绑定的队列



## 2: 交换机的属性:

Name: 交换机的名称

Type: 交换机的类型, direct, topic, fanout, headers

Durability: 是否需要持久化

autodelete: 加入没有队列绑定到该交换机, 那么该交换机会自动删除

Internal: 当前交换机是否用户rabbitmq内部使用不常用, 默认为false

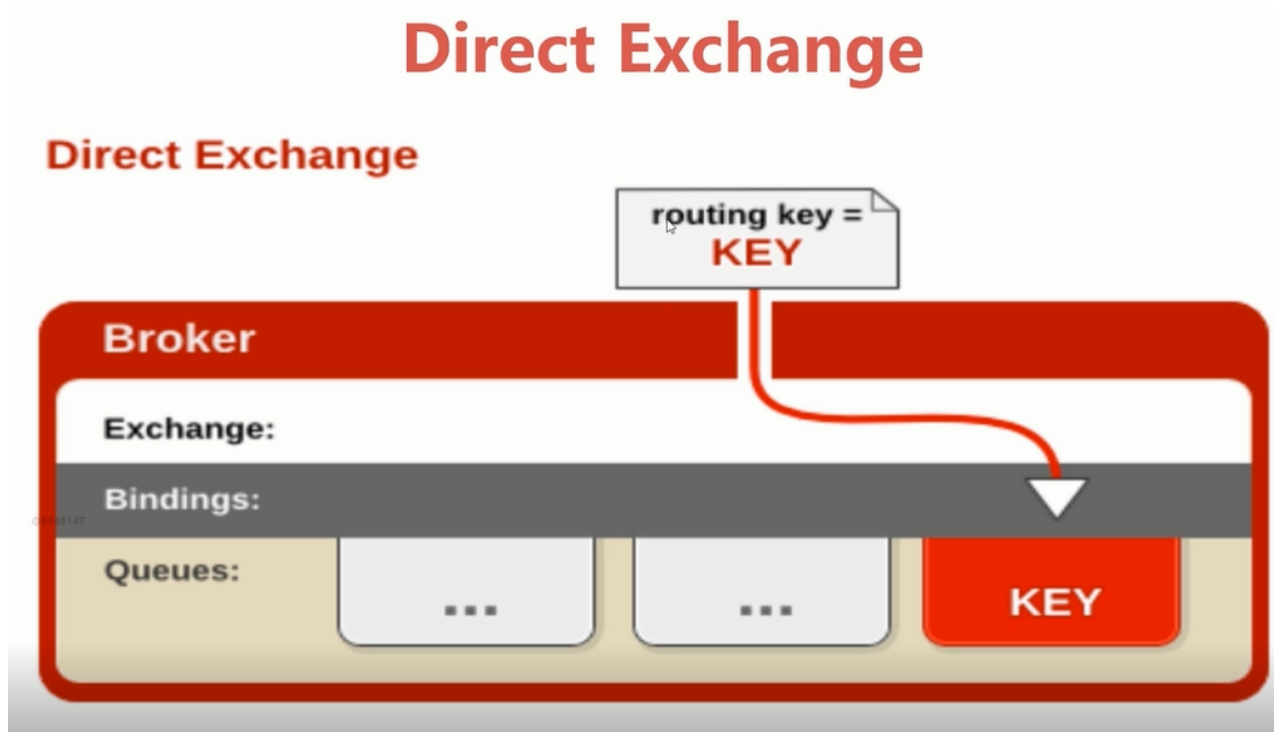
Argurements: 扩展参数, 用户扩展AMQP 定制化协议

## 3: 交换机的类型

3.1) 直连交换机: direct exchange

所以发送的direct exhchange 的消息都会被投递到与routekey名称(与队列名称)相同的queue上

**\*\*direct模式下，可以使用rabbitmq自定exchange----> default exchange 所以不需要交换机和任何队列绑定，消息将会投递到route\_key名称和队列名称相同的队列上**



代码演示:

直连交换机生产者

```
public static void main(String[] args) throws IOException, TimeoutException {
    ConnectionFactory connectionFactory = new ConnectionFactory();
    connectionFactory.setHost("47.104.128.12");
    connectionFactory.setPort(5672);
    connectionFactory.setVirtualHost("cloudmall");
    Connection connection = connectionFactory.newConnection();
    Channel channel = connection.createChannel();
    String directExchangeName = "test.direct.exchange";
    String routingKey = "test.direct.key";
    channel.basicPublish(directExchangeName, routingKey, null, "测试直连交换机。。。。。。".getBytes());
}
```

直接交换机消费者

```
public static void main(String[] args) throws IOException, TimeoutException, InterruptedException {
    ConnectionFactory connectionFactory = new ConnectionFactory();
    connectionFactory.setHost("47.104.128.12");
    connectionFactory.setPort(5672);
    connectionFactory.setVirtualHost("cloudmall");
    Connection connection = connectionFactory.newConnection();
    Channel channel = connection.createChannel();
```

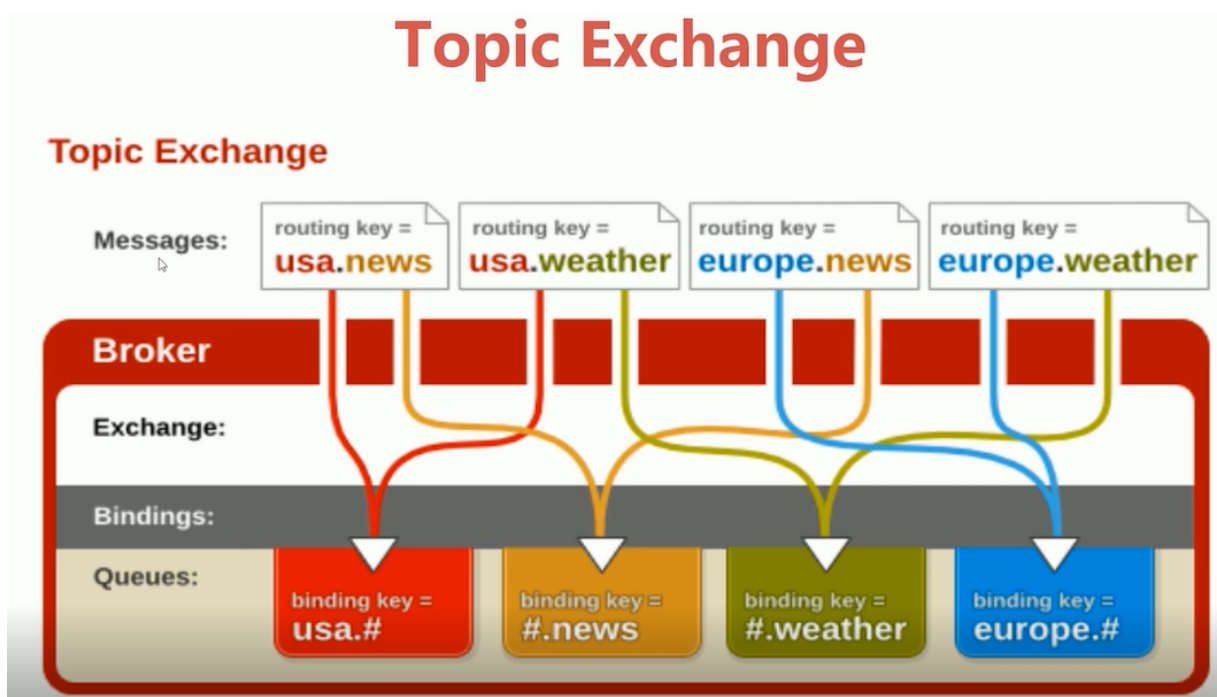
```
//声明一个交换机
channel.exchangeDeclare("test.direct.exchange","direct",true,false,false,null);
//声明一个队列
channel.queueDeclare("test.direct.queue",false,false,false,null);
//队列绑定到指定的交换机上
channel.queueBind("test.direct.queue","test.direct.exchange","test.direct.key");
QueueingConsumer queueingConsumer = new QueueingConsumer(channel);
channel.basicConsume("test.direct.queue",true,queueingConsumer);
QueueingConsumer.Delivery delivery = queueingConsumer.nextDelivery();
System.out.println(new String(delivery.getBody()));
System.out.println(delivery.getEnvelope().getExchange());
System.out.println(delivery.getEnvelope().getRoutingKey());
}
```

### 3.2)主题交换机 TopicExchange

就是在队列上绑定到topic 交换机上的路由key 可以通过通配符来匹配的通配符的规则是

比如: log.# : 可以匹配一个单词 也可以匹配多个单词 比如 log.# 可以匹配log.a log.a.b log.a.b

log.\* 可以匹配一个单词 比如 log.\* 可以匹配log.a 但是不可以匹配log.a.b



代码演示 :topic exchange 生产者

```

public static void main(String[] args) throws IOException, TimeoutException {
    ConnectionFactory connectionFactory = new ConnectionFactory();
    connectionFactory.setVirtualHost("cloudmall");
    connectionFactory.setHost("47.104.128.12");
    connectionFactory.setPort(5672);
    Connection connection = connectionFactory.newConnection();
    Channel channel = connection.createChannel();
    String topExchangeName = "top.exchange";
    String routingKey1 = "top.key.1";
    String routingKey2 = "top.key.2";
    channel.basicPublish(topExchangeName,routingKey1,null,"测试交换机".getBytes());
    channel.basicPublish(topExchangeName,routingKey2,null,"测试交换机".getBytes());
    channel.close();
    connection.close();
}

```

### 代码演示: topic exchange 消费者

```

public class Top4Consumer {
    public static void main(String[] args) throws IOException, TimeoutException, InterruptedException {
        ConnectionFactory connectionFactory = new ConnectionFactory();
        connectionFactory.setVirtualHost("cloudmall");
        connectionFactory.setHost("47.104.128.12");
        connectionFactory.setPort(5672);
        Connection connection = connectionFactory.newConnection();
        Channel channel = connection.createChannel();
        String topicExchangeName = "top.exchange";
        //声明一个交换机
        channel.exchangeDeclare(topicExchangeName,"topic",true,true,false,null);
        //声明一个队列
        channel.queueDeclare("top.queue",true,false,true,null);
        //队列绑定到交换机
        channel.queueBind("top.queue",topicExchangeName,"top.#");
        QueueingConsumer queueingConsumer = new QueueingConsumer(channel);
        //参数: 队列名称、是否自动ACK、Consumer
        channel.basicConsume("top.queue", true, queueingConsumer);
        while (true) {
            QueueingConsumer.Delivery delivery = queueingConsumer.nextDelivery();
            System.out.println(new String(delivery.getBody()));
        }
    }
}

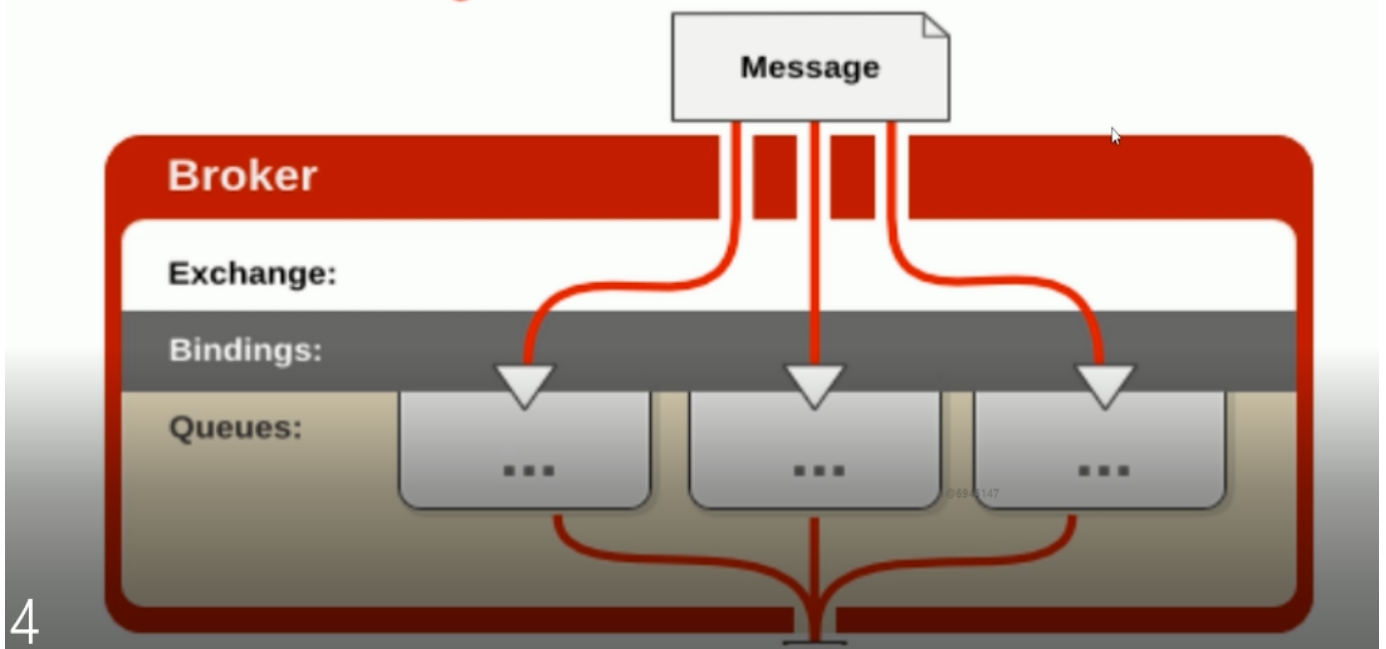
```

### 3.3) 扇形交换机(fanout exchange)

就是消息通过从交换机到队列上不会通过路由key 所以该模式的速度是最快的 只要和交换机绑定的那么消息就会被分发到与之绑定的队列上

# Fanout Exchange

## Fanout Exchange



### 代码演示 扇形交换机 模式下的生产着

```
public static void main(String[] args) throws IOException, TimeoutException {
    ConnectionFactory connectionFactory = new ConnectionFactory();
    connectionFactory.setHost("47.104.128.12");
    connectionFactory.setPort(5672);
    connectionFactory.setVirtualHost("cloudmall");
    Connection connection = connectionFactory.newConnection();
    Channel channel = connection.createChannel();
    String fanoutExchangeName = "test.fanout.exchange";
    String routingKey = "test.fanout.key";
    channel.basicPublish(fanoutExchangeName, routingKey, null, "測試扇形交换机。 . . . . ".getBytes());
}
```

### 扇形交换机模式下的消费者

```
public static void main(String[] args) throws IOException, TimeoutException, InterruptedException {
    ConnectionFactory connectionFactory = new ConnectionFactory();
    connectionFactory.setHost("47.104.128.12");
    connectionFactory.setPort(5672);
    connectionFactory.setVirtualHost("cloudmall");
    Connection connection = connectionFactory.newConnection();
    Channel channel = connection.createChannel();
```

```
//创建交换机
channel.exchangeDeclare("test.fanout.exchange","fanout",true,true,false,null);
//创建队列
channel.queueDeclare("test.fanout.queue",true,false,true,null);
//绑定队列
channel.queueBind("test.fanout.queue","test.fanout.exchange","");
QueueingConsumer queueingConsumer = new QueueingConsumer(channel);
channel.basicConsume("test.fanout.queue",true,queueingConsumer);
while (true) {
    QueueingConsumer.Delivery delivery = queueingConsumer.nextDelivery();
    System.out.println(new String(delivery.getBody()));
}
```

```
}
```

## 十一: 队列,绑定虚拟主机, 消息

**绑定:** exchange 与之间的连接关系(通过路由规则)

**队列:**用来存储消息的实体

**队列的属性:** durability 消息是否被持久化

AutoDelete :表示最后一个监听被移除那么该队列就会被删除

**消息:**用来生产者和消费者之间传递数据的

**消息属性:** 包括消息体body 和属性 properties

**常用属性:**delivery mode , headers, content\_type(消息类型) content\_encoding(消息编码),priority(消息优先级)

correlation\_id(最为消息唯一的id),reply\_to (消息失败做重回队列) ,expiration(消息的过期时间),message\_id(消息id);

timestamp,type,user\_id , app\_id,cluster\_id等

**自定义消息属性的消费端代码:**

```
public static void main(String[] args) throws IOException, TimeoutException, InterruptedException {
    ConnectionFactory connectionFactory = new ConnectionFactory();
    connectionFactory.setHost("47.104.128.12");
    connectionFactory.setPort(5672);
    connectionFactory.setVirtualHost("/");
    //2 通过连接工厂创建连接
    Connection connection = connectionFactory.newConnection();
    //3 通过connection创建一个Channel
    Channel channel = connection.createChannel();
    //4 声明 (创建) 一个队列
    String queueName = "test001";
```



```

channel.queueDeclare(queueName, true, false, false, null);
//5 创建消费者
QueueingConsumer queueingConsumer = new QueueingConsumer(channel);
//6 设置Channel
channel.basicConsume(queueName, true, queueingConsumer);
while(true){
    //7 获取消息
    Delivery delivery = queueingConsumer.nextDelivery();
    String msg = new String(delivery.getBody());
    System.out.println("消费端: " + msg);
    System.out.println(delivery.getProperties());
    System.out.println(delivery.getProperties().getHeaders());
}
}

```

## 自定义消息属性的生产者

```

public static void main(String[] args) throws IOException, TimeoutException {
    //创建连接工厂
    ConnectionFactory connectionFactory = new ConnectionFactory();
    connectionFactory.setHost("47.104.128.12");
    connectionFactory.setPort(5672);
    connectionFactory.setVirtualHost("/");
    //创建连接
    Connection connection = connectionFactory.newConnection();
    //创建channel
    Channel channel = connection.createChannel();
    Map<String, Object> extraMap = new HashMap<>();
    extraMap.put("k1", "v1");
    extraMap.put("k2", "v2");
    /**
     * 附带额外信息的信息体
     */
    AMQP.BasicProperties basicProperties = new AMQP.BasicProperties.Builder()
        .deliveryMode(2) //2为持久化, 1 不是持久化
        .appId("测试appid")
        .clusterId("测试集群id")
        .contentType("application/json")
        .contentEncoding("UTF-8")
        .headers(extraMap).build();
    for (int i = 0; i < 10; i++) {
        String targetMsg = "这是我的第【" + (i + 1) + "】条消息";
        channel.basicPublish("", "test001", basicProperties, targetMsg.getBytes());
    }
    channel.close();
    connection.close();
}

```



